Multi-GPU Implementation of the Numerical Algorithm Enhancing Discrete Element Modeling Performance

V. D. Chepelenkova(✉), D. I. Prokhorov, and V. V. Lisitsa

Sobolev Institute of Mathematics SB RAS, Novosibirsk, Russia
vchepelenkova@gmail.com

Abstract. This paper presents an algorithm to combine GPU computations for the two-dimensional discrete element method (DEM) with MPI data transfers between computer cluster nodes in distributed-memory systems for one-dimensional domain decomposition. The proposed algorithm redistributes particles between processes, deleting them or adding to the list of particles inside a current subdomain, respectively, in parallel with main GPU calculations to decrease the time loss of calling transferring functions. Strong and weak scaling tests were conducted for the implemented algorithm to show its effectiveness in adding more computational resources to the fixed-size problem and increasing the problem size according to the allocated resources.

Keywords: Multi-GPU · MPI data transfers · Discrete element method

1 Introduction

Modeling crack propagation or contact problems in solid media can be classified as problems with discontinuous properties of a material arising in rock geophysics or any other applications to material science. Continuum-based methods, e.g., finite difference or finite element methods, are the most conventional approach to obtaining a numerical solution in such cases and are widely used for modeling elastoplastic materials. The key feature is in coming down to solving a system of linear equations, which can be easily implemented and parallelized in most cases. However, as reflected in the name of this class of methods, its governing equations are based on principles of continuum mechanics. Therefore, they have common limitations when used to solve discontinuous problems. Firstly, potentially contacting surfaces or the initial placement of a crack must be defined before the simulation starts, and secondly, mesh parts both for contacting surfaces and for fracture lines are required to be compatible to form a continuous mesh.

On the other hand, particle methods, especially the discrete element method, presented in [1], are not based on continuum mechanics, which makes it able to

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15888, pp. 317–329, 2026.
https://doi.org/10.1007/978-3-031-97596-7_21

naturally form contacts and fractures without specifying any extra information beforehand. The reason for this is that solid media in DEM is considered a system of small spherical particles bound with elastic springs that can break, thus forming micro-cracks. However, the major drawback in using DEM is its computational intensity: the given medium has to be discretized to a significant number of particles with their elastic properties for each one (memory intensive aspect), which leads to a considerable amount of time-consuming operations when calculating particle pairwise interactive forces within a specific range (time intensive aspect). The way to overcome the aforementioned issues is to use graphics accelerators for more efficient computation and multiple cluster nodes as a distributed memory system exchanging data via MPI. More details on comparing continuum-based and particle methods are given in [2,3].

This paper presents an algorithm to effectively combine multiple-GPU computations for a two-dimensional DEM model that simulates uniaxial tests on porous samples, with MPI data transfers for one-dimensional domain decomposition. Strong and weak scaling tests were conducted to more precisely analyze the implemented program's effectiveness.

2 Discrete Element Modeling

DEM simulations of solid and granular materials are based on a representation of the given medium as a set of spherical (or circular in 2D space) particles interacting by the laws of Newtonian mechanics, so governing equations for particle motion are expressed as

$$m_i \frac{d^2 \boldsymbol{r}_i}{dt^2} = \boldsymbol{f}_i + m_i \boldsymbol{g} - \gamma \boldsymbol{v_i}, \qquad i = 1, ...N \tag{1}$$

where $\boldsymbol{f}_i = \sum_{j=1}^{N} \boldsymbol{f}_{ij}$ is a total force acting on i-th particle, which is a sum of all pairwise interactions, $\boldsymbol{g}$ is a gravitational constant, and N is the number of particles in a given sample. A term $\gamma \boldsymbol{v}_i$ stands for a dissipative force, the amplitude of which can be altered by changing the value or variable γ. This force is intended to slow particle motion and reach an equilibrium state.

Contacting particles are assumed to have a non-empty yet small overlap region compared to their radii. For instance, if particles i and j have positions $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ with radii R_i and R_j respectively, the overlap size between them is calculated using formula $\delta = R_i + R_j - \|\boldsymbol{r}_i - \boldsymbol{r}_j\|$. Therefore, we can define a normal to the contact plane vector

$$\boldsymbol{n}_{ij} = \frac{\Delta \boldsymbol{r}_{ij}}{\|\Delta \boldsymbol{r}_{ij}\|}, \qquad \Delta \boldsymbol{r}_{ij} = \|\boldsymbol{r}_i - \boldsymbol{r}_j\|,$$

allowing us to decompose a force of pairwise interactions to $\boldsymbol{f}_{ij} = f_{ij}^n \boldsymbol{n}_{ij} + \boldsymbol{f}_{ij}^t$ (Fig. 1). For ease of notation, indices ij are hereafter omitted.

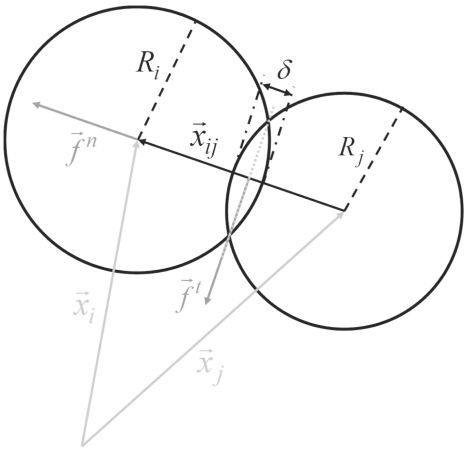


Fig. 1. Normal and tangential components of $\boldsymbol{f}_{ij}$.

There exist multiple force models based on a principle that overlap region is regarded as an elastic spring with stiffness k, connecting two particles [4,5]. In this paper, a linear one is chosen, so for a model used in all further computations, f^n and $\boldsymbol{f}^t$ are defined as follows:

$$f^n = \begin{cases} k\delta, & \delta > 0, \\ -k|\delta|, & -d_{max} < \delta < 0, \\ 0, & \delta < -d_{max}, \end{cases} \qquad \boldsymbol{f}^t = \begin{cases} \boldsymbol{f}_0^t, & \|\boldsymbol{\xi}\| < d_{max}, \\ \mu f^n \dfrac{\boldsymbol{f}_0^t}{\|\boldsymbol{f}_0^t\|}, & \|\boldsymbol{\xi}\| \geq d_{max}, \end{cases}$$

where k is the aforementioned stiffness coefficient, μ is a friction coefficient, ξ is a relative displacement of two particles along their contact plane, and $\boldsymbol{f}_0^t = -k^t\boldsymbol{\xi}$ with stiffness k^t. It is assumed that $\boldsymbol{\xi} = 0$ when a new contact is established, then after each time step τ it is updated according to formula $\boldsymbol{\xi}^{(n+1)} = \boldsymbol{\xi}^{(n)} + \tau\boldsymbol{v}^t$. More details on the definition of $\boldsymbol{f}^t$ are given in [6]. Parameter d_{crit} stands for the maximum possible distance particles can move away from each other before the bond between them breaks. From later on, only a repulsive part of f_n can arise, meaning that $f^n = 0$ for $\delta < 0$, and $\boldsymbol{f}^t$ for any contact is limited to sliding friction only, so $f^t = \mu f^n$ for $\delta > 0$.

3 Numerical Approximation

To keep track of i-th particle position ($i = 1, \ldots, N$) at each time step, we use the Velocity Verlet integration of Eq. 1, which has the form

$$\begin{aligned} \boldsymbol{r}_i^{n+1} &= \boldsymbol{r}_i^n + \tau \boldsymbol{v}_i^n + \frac{\tau^2}{2} \frac{\boldsymbol{f}_i(\boldsymbol{r}^n, \boldsymbol{v}^n)}{m_i}, \\ \boldsymbol{v}_i^{n+\frac{1}{2}} &= \boldsymbol{v}_i^n + \frac{\tau}{2} \frac{\boldsymbol{f}_i(\boldsymbol{r}^n, \boldsymbol{v}^n)}{m_i}, \\ \boldsymbol{v}_i^{n+1} &= \boldsymbol{v}_i^{n+\frac{1}{2}} + \frac{\tau}{2} \frac{\boldsymbol{f}_i(\boldsymbol{r}^{n+1}, \boldsymbol{v}^{n+\frac{1}{2}})}{m_i}, \end{aligned} \tag{2}$$

where m_i denotes i-th particle's position.

The stability criterion for 2 derives from the assumption that any particle can move no further than a distance from the other particle in one time step. It is proposed in [7] that such a condition can be expressed as

$$\tau \leq 0.2 D_{min} \sqrt{\frac{\rho}{K + 4/3G}}, \tag{3}$$

where D_{min} is the smallest particle diameter, K and G denote a given medium's bulk and shear moduli, and ρ stands for density.

3.1 Optimization of Inter-Particle Force Calculations

The most straightforward way to calculate the total force for each particle is to iterate over all the others. This approach requires N^2 operations, with most of them being redundant since particles placed far away from each other do not interact. Hence, some optimization is needed to limit the area of calculations for each particle.

For this reason, we define a uniform rectangular grid built by the criterion in (3), which means that a particle can move no further than to an adjacent cell in one time step. To do so, let us define a grid cell as

$$C_{kl} = \{(x, y) \in \mathbb{R}^2 \mid kh \leq x \leq (k+1)h,\ lh \leq y \leq (l+1)h\},$$

where the cell size $h = 2R_{max} + d_{max}$ does meet the condition above. A particle i is said to belong to cell C_{kl} if $\boldsymbol{r}_i \in C_k l$. Therefore, for a particle i all possible interactions can only occur inside a domain $J_i = \{C_{k'l'} | k' \in \{k-1, k, k+1\},\ l' \in \{l-1, l, l+1\}\}$, and an expression $\boldsymbol{f}_i = \sum_{j=1}^{N} \boldsymbol{f}_{ij}$ can be reduced to $\boldsymbol{f}_i = \sum_{j \in J_i} \boldsymbol{f}_{ij}$, which is much less time consuming than complete enumeration.

It is worth mentioning that particles can move in and out of a cell, leading to the necessity of updating corresponding data at each time step, which is, nevertheless, still more efficient than the naive approach if executed on GPU.

4 Implementation of the Algorithm

The execution process of the implemented algorithm contains two major stages. The former generates a sample filled with tightly packed particles and forms bonds between them to create a solid body with given elastic parameters. More details on this stage execution are provided in Sect. 4.1. The latter is a stage of simulating the time-wise evolution of sample characteristics under a specific type of loading.

A flowchart in Fig. 2 presents a brief overview of a modeling stage. All operations inside a loop are executed on GPU without transferring data to CPU and back except occasional writing to file. A stress or strain tensor is measured by averaging all individual particle values over the total sample volume. The only part operating on both CPU and GPU is the subprogram calculating particle positions and velocities, where MPI data transfers are performed in parallel with GPU calculations for the next time step. This process is explained below, in Sect. 4.2.

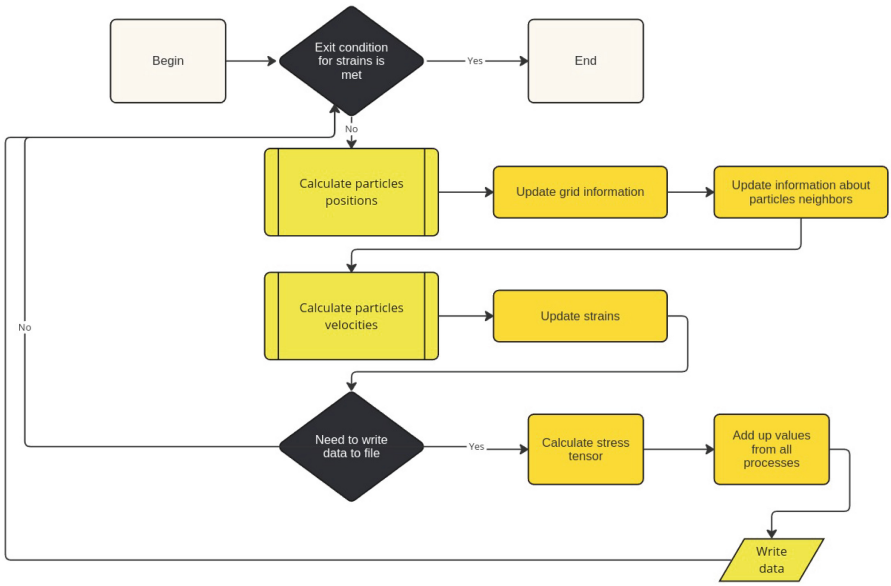


Fig. 2. Flowchart of the implemented algorithm.

Nevertheless, using CUDA and MPI for DEM simulations has some peculiarities. One arises from a particle structure, which contains both value types and pointers to array data stored in other memory locations and, thus, not aligned with the parent structure as shown in Fig. 3. Moreover, all particles in a model are collected in an array, which leads to the problem of transferring data both to

GPU and to another cluster node and back. The most straightforward approach is to send and receive all particles with all their fields separately, sending inner array data instead of pointers. Such a way is the most inefficient one since every transferring operation, either with GPU or with another node, is exceptionally time-consuming and, thus, should be performed as few times as possible.

Representing an array of structures with a sequence of bytes arranged in a certain order, also called serialization, reduces the number of individual calls of transferring functions to just once every time the list particles should be copied to other memory space. The default approach to serialization is to cast a structure to an array of bytes, which is not viable from the point of sending data to a separate memory location because default casting to bytes only preserves pointers to an exact position in RAM and not the values it points to. Thus, after transferring to any other physical device, not only do array data become absent, but all pointers, when deserialized, turn out invalid in the current memory space and lead to a runtime error if dereferenced. In order to avoid this issue, we use deep serialization, which is performed by switching every single particle's pointers to the data they pointed at and aligning them to a byte sequence. A total byte array to be sent consists of byte sequences for all particles placed sequentially together.

It is worth noting that the deserialization process is primarily the opposite of serialization, meaning that for an input byte array using the known data about particle contents and number of particles, we can restore data as a similar array of structures located at another independent memory. For GPU computations, each particle to deserialize can be assigned to a single thread, decreasing the execution time.

The other issue is the requirement to keep track of particles moved through process boundaries on each iteration, which is more complex than a similar exchange in the finite difference method. Section 4.2 provides more details on handling MPI-exchanges for DEM.

4.1 Packing Stage

To obtain a set of tightly packed particles, we use the combination of two methods: generating and moving particles, filling a given domain (rectangular in our case) with particles placed in such a way that every particle has at least two contact points with either a wall or another particle. Figure 4 shows an example of packing for a small amount of particles.

The first step is to generate enough particles to fill the entire domain. To do so, we use Poisson Disk Sampling [8], the general idea of which is to randomly generate a uniformly distributed set of points in such a way that a distance between each pair of them is no less than a particular input value k. In order to ensure a non-zero distance between particles, the value $k = 2R_{max}$ was chosen. After that, each point is used as a center of a circle with radius $r \in 0.1(R_{min},\ R_{max})$ picked randomly as well, forming an initial position and size of the particle.

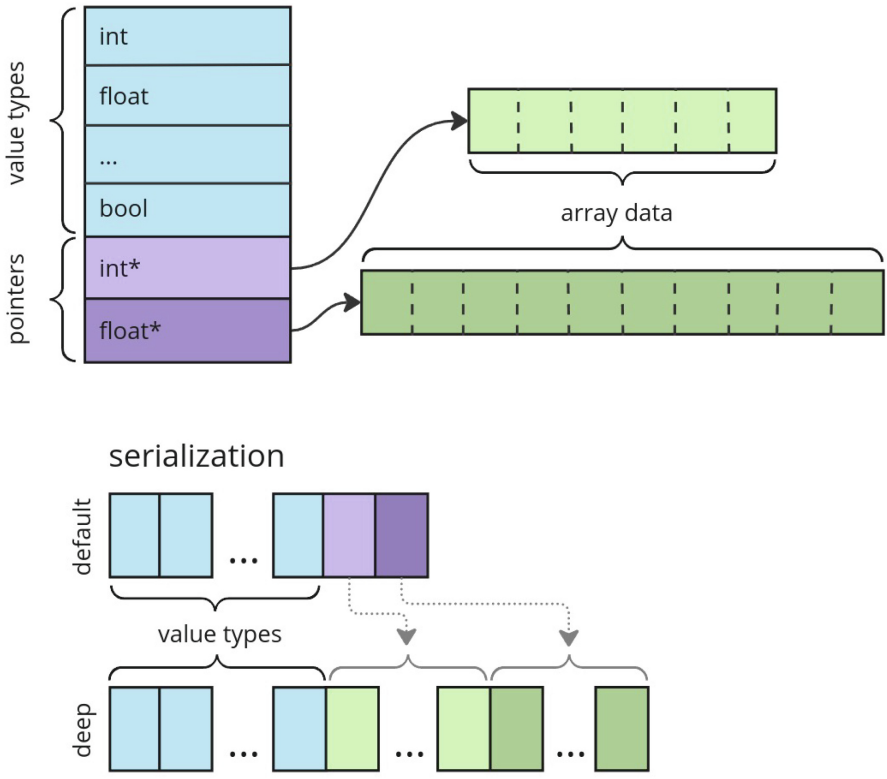


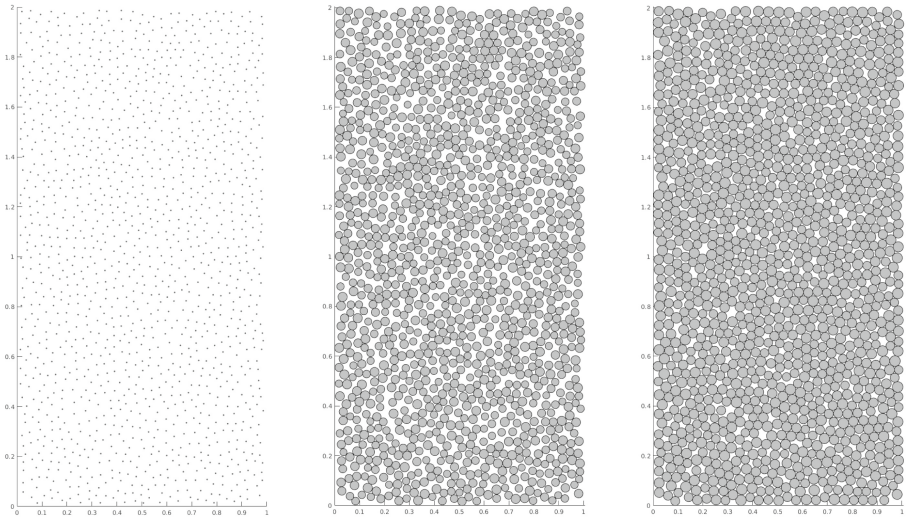
Fig. 3. On the upper part: schematic representation of an individual particle structure. On the lower part: two ways of serialization compared.

The second step is based on a Lubachevsky-Stillinger algorithm [9], which results in densely packed objects within the given domain. The central concept of the algorithm is to use event-driven simulation, which, contrary to time-driven simulations, steps over time not with a fixed time step but with a period from the instant moment to the next closed event.

The process starts with assigning a random velocity $\boldsymbol{v}$ with components in range $(0,\ 1)$ to each particle. An event, in this case, is considered to be either the collision of two particles or the collision of a particle with any wall.

The major steps of the algorithm are as follows:

1. Predicting the next closest event, obtain the time Δt it will happen, and a pair of particles (or a particle and a wall) taking part in it.
2. Moving particles to the distance defined as $\Delta \boldsymbol{r} = \boldsymbol{v}\Delta t$.
3. Performing the event, i.e., changing particle velocities concerning laws of elastic collision.



(a) (b) (c)

Fig. 4. Packing stages: Output from Poisson Disk Sampling (a), packing with Lubachevsky-Stillinger algorithm in progress (b), ready sample (c).

4. Increase the radius of each particle according to $R^{n+1} = \alpha R^n$, where α is an input parameter defining a particle growth rate.

The process repeats while Δt remains above a certain threshold, meaning that time between events becomes negligible when particles are close enough to treat them as packed and stop the algorithm.

4.2 MPI-Exchanges

Discrete element element is based on a highly memory-intensive algorithm as it requires a significant number of particles to handle a representative sample. At the same time, video random-access memory (VRAM) is limited compared to RAM. A possible solution to this problem is splitting the domain into smaller parts and distributing them to multiple GPUs or nodes of a computer cluster.

Every process has its subgrid with corresponding particles and one row from each adjacent process as halo cells to use information from other processes. In this work, we use 1D decomposition so that each MPI process has only two (one for the first or the last one) rows of halo cells, as depicted in Fig. 5a.

The major peculiarity of using MPI for DEM is the necessity to keep track of particles moving through process boundaries on each iteration, which becomes a more complex task than exchanging data in the finite difference method. In total, there are three possible cases for a particle to move (Fig. 5b):

– Particle stayed in a current subdomain.

– Particle moved into the current subdomain and must be handled by the current process.
– Particle moved out of the current subdomain and should be deleted from the list of particles handled by the process

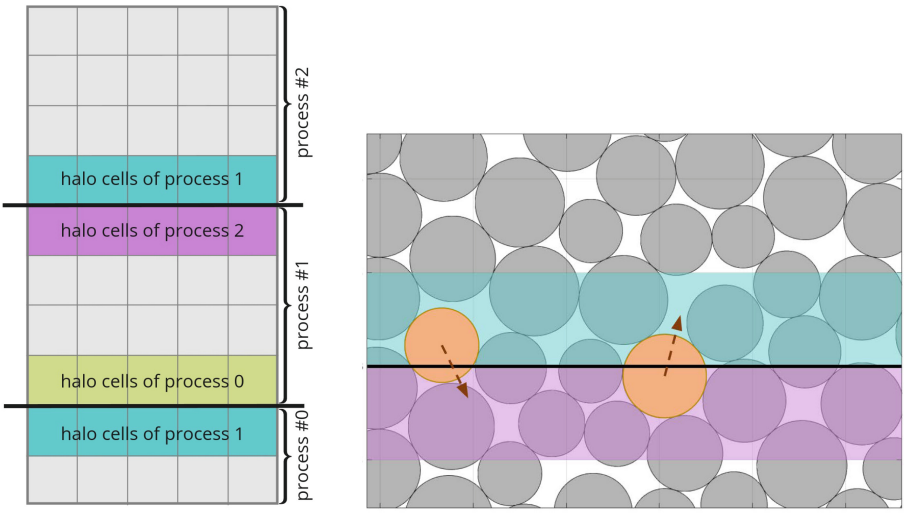


Fig. 5. Example of a domain decomposition for each process (a), possible particle movements to be handled separately (b).

To take into account all of the above cases, we implemented the following algorithm (Fig. 6):

1. Calculate positions and velocities for particles in halo cells.
2. For each adjacent process, if particles are moved out of the current subdomain, fill an array with their indices.
3. Transfer them to RAM as well as the first and/or the last row of the subgrid in order to send them as halo cells for adjacent processes
4. Send the first and/or the last rows via MPI to adjacent processes
5. Receive similar arrays from adjacent processes.
6. If there exist particles moved through the boundary: send and/or receive corresponding arrays prepared in step 2.
7. If particles moved into the current subdomain, insert them into the list of particles handled by the current process and delete them from particles in halo cells
8. If particles move out to an adjacent process, delete them from the list of particles handled by the current process.

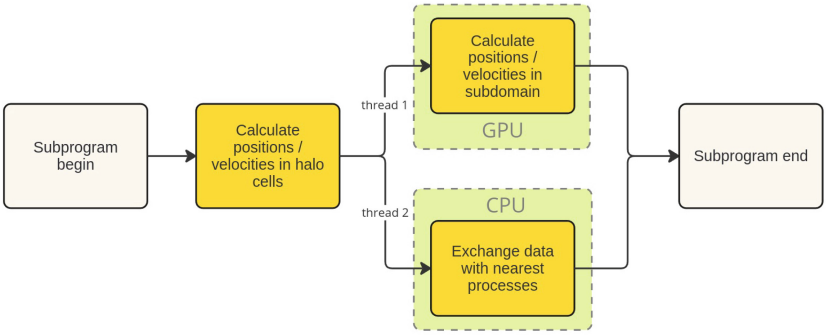


Fig. 6. Subprogram for executing both calculations and data exchange in parallel.

9. Transfer received and updated halo arrays back to VRAM.

Steps 3-9 are executed on CPU, while positions and velocities for all particles inside a subgrid, except halo cells, are calculated on GPU in parallel.

5 Scalability Analysis

Any parallel program using computations on a distributed memory system can be tested for strong and weak scaling. The former shows its capability to gain more efficiency when more resources are added to the system, which can be estimated using the formula

$$E_{strong} = \frac{t_1}{Nt_N},$$

where t_N is the execution time measured for a program on N processes. In a perfect case, every process uses all its resources, and the whole program works strictly in parallel, which gives $E_{strong} = 1$. However, in reality, there is almost no way to avoid at least some parts of serial code being present since processes have to transfer data to each other, which has to be synchronized at some point. Moreover, there exists a bottleneck when adding more processes to the system, leading to a considerable decrease in efficiency. The reason for this is that the amount of data exchanged between processes significantly exceeds the amount of calculations made by each individual process. If this state is reached, it makes no sense to keep adding resources as it will not speed up the execution time.

Figure 7a shows the result of a strong scaling test of the implemented algorithm ran on 1, 2, 4, 8, and 16 MPI processes with CPUs Intel Xeon CPU E5-2697 and GPUs NVIDIA Tesla K40. Each sample contained 112819 particles. It can be seen that efficiency reaches the value of about 0.8 at 8 processes, and after that, it rapidly decreases, meaning that for the given sample size, no speedup would be gained for more than cluster nodes.

Week scaling, in its turn, is measured as

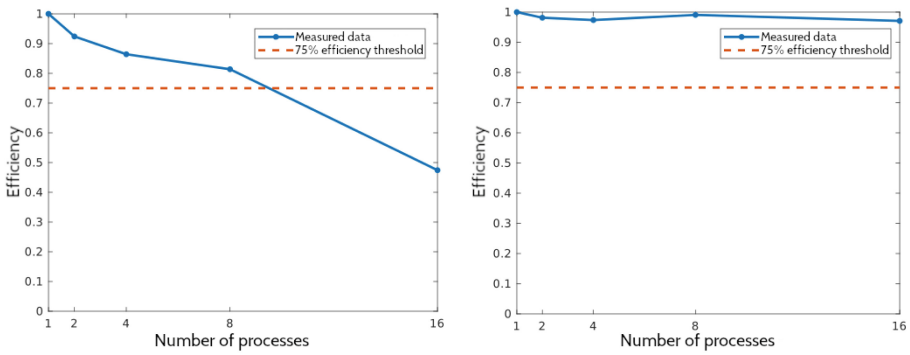
$$E_{weak} = \frac{t_1}{t_N},$$

for a series of problems, their size increased so that the amount of work remains intact for every process. E_{weak} indicates how much the serial part of the code is affected by increasing the total amount of work. In a perfect case, it should be at $E_{weak} = 1$, meaning that data transfers and other serial parts do not take more time concerning the problem size.

Figure 7b illustrates the weak scaling test results for samples with their size increasing, as shown in Table 1. It is evident that the efficiency of the implemented algorithm mostly remains constant for any number of processes and is experiencing negligible changes in value, which is probably related to the fact that, due to the randomized particle generation based on Poisson disk Sampling, it seems impossible to achieve the exact multiple of the number of particles in a scaled domain. However, the overall trend leads to the conclusion that the algorithm has little dependence on the problem size.

Table 1. Domain size and the number of particles for weak scaling.

N processes	1	2	4	8	16
Domain size, mm × mm	6 × 12	6 × 24	6 × 48	6 × 96	6 × 192
Number of particles	112 827	225 699	451 431	902 108	1 804 499



(a) (b)

Fig. 7. Results of strong (a) and weak (b) scaling tests.

6 Conclusion

The discrete element method is a preferable approach to model solid media experiencing discontinuities of any sort, e.g., crack forming and propagation or contact problems, than continuum-based methods since it avoids their major drawbacks as the necessity to point out the initial state of a crack and treat adjacent nodes in a specific way to correctly bind discontinuities with governing equations based on continuum mechanics assumptions. However, for DEM simulations, a given sample is discretized as a set of particles, which should be numerous enough to provide correct results, which makes this method rather demanding both in time and memory resources.

In this paper, we proposed an algorithm combining data exchanges between computer cluster nodes in distributed-memory systems with GPU computations. The former allows to overcome both RAM and VRAM limitations of a single node, while the latter is a widely known way to significantly reduce the execution time of a program containing a large number of identical operations, also known as SIMD (Single Instruction, Multiple Data).

The principal peculiarity when performing DEM simulations is not only to transfer via MPI effectively but to keep track of particles moved through a boundary between adjacent processes. The proposed algorithm redistributes particles between processes, deleting them or adding to the list of particles inside a current subdomain, respectively, in parallel with GPU calculations to decrease the time loss of calling transferring functions.

Strong and weak scaling test analyses were conducted for the implemented algorithm, showing its effectiveness in adding more computational resources to the fixed-size problem and increasing the problem size according to the allocated resources. The decrease in strong scaling effectiveness with an increasing number of nodes is a typical behavior interpreted as a certain threshold, beyond which the size problem for each process makes GPU computations take less time to overlap data transfers. This effect is inevitable for any program using MPI and indicates the maximum amount of resources to handle the given problem size effectively.

Acknowledgments. This study was funded by the Russian Science Foundation (grant no. 19-77-20004-П).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Cundall, P.A., Strack, O.: A discrete numerical model for granular assemblies. Geotechnique **29**(1), 47–65 (1979)
2. Gray, G.G., Morgan, J.K., Sanz, P.: Overview of continuum and particle dynamics methods for mechanical modeling of contractional geologic structures. J. Struct. Geol. **59**, 19–36 (2014)

3. Lisjak, A., Grasselli, G.: A review of discrete modeling techniques for fracturing processes in discontinuous rock masses. J. Rock Mech. Geotech. Eng. **6**(4), 301–314 (2014)
4. Chen, F., Xia, Y., Klinger, J.L., Chen, Q.: A set of hysteretic nonlinear contact models for DEM: theory, formulation, and application for lignocellulosic biomass. Powder Technol. **399**, 117100 (2022)
5. Yan, Z., Wilkinson, S.K., Stitt, E.H., Marigo, M.: Discrete element modelling (DEM) input parameters: understanding their impact on model predictions using statistical analysis. Comput. Part. Mech. **2**(3), 283–299 (2015). https://doi.org/10.1007/s40571-015-0056-5
6. Luding, S.: Introduction to discrete element methods: basic of contact force models and how to perform the micro-macro transition to continuum theory. Eur. J. Environ. Civ. Eng. **12**(7–8), 785–826 (2008)
7. Hardy, S., McClay, K., Munoz, J.A.: Deformation and fault activity in space and time in high-resolution numerical models of doubly vergent thrust wedges. Mar. Pet. Geol. **26**(2), 232–248 (2009)
8. Bridson, R.: Fast Poisson disk sampling in arbitrary dimensions. SIGGRAPH Sketches **10**(1), 1–1 (2007)
9. Lubachevsky, B.D., Stillinger, F.H.: Geometric properties of random disk packings. J. Stat. Phys. **60**, 561–583 (1990)